Rozwiązania zadań z Mistrzostw Polski Szkół Średnich w Programowaniu Zespołowym 2023

Zadanie A

W zadaniu mamy dane K złotych monet z których każda dzieli się na G groszy. Należy podzielić tę kwotę na N osób w taki sposób aby suma N części zaokrąglonych do pełnych monet była jak najmniejsza (chcemy zmaksymalizować kwotę "zaoszczędzoną" przy zaokrąglaniu). Przyjmujemy tutaj naturalne kryterium zaokrąglania, czyli jeśli groszowa końcówka jest równa przynajmniej $\frac{G}{2}$, zaokrąglamy w górę, a w przeciwnym przypadku w dół.

Bez straty ogólności możemy założyć, że przynajmniej $N-1$ części będzie mniejsze niż pełna moneta. Zauważmy, że jeśli część groszowa będzie równa co najwyżej $\lceil \frac{G}{2} \rceil - 1$, to kwota będzie zaokrąglona w dół. Na dodatek jest to maksymalne możliwe zmniejszenie wyniku – w ten sposób "wygrywamy" $\lceil \frac{G}{2} \rceil - 1$ groszy. Dlatego jedną z optymalnych strategii jest przydzielanie kolejnym osobom (maksymalnie $N-1$) $\lceil \frac{G}{2} \rceil - 1$ groszy dopóki nie osiągniemy K pełnych złotych monet. N-tej osobie przydzielamy pozostałą kwotę. Jeśli K-monet to za mało, żeby każdej z $N-1$ pierwszych osób przydzielić $\lceil \frac{G}{2} \rceil - 1$ groszy, kolejnej osobie przyznajemy pozostałą kwotę, a całej reszcie nie dajemy nic.

Takie rozwiązanie jest optymalne. Jeśli N-ta osoba otrzymała nie więcej niż $\lceil \frac{G}{2} \rceil - 1$ groszy, to każda kwota została zaokrąglona do zera, więc wynik w oczywisty sposób jest optymalny. Rozważmy teraz przeciwny przypadek. Po pierwsze, zaoszczędzoną kwotę zawsze da się wyrazić jako pewną liczbę pełnych złotych monet. Po drugie, liczba ta nie może przekroczyć $N \cdot \lceil \frac{G}{2} \rceil - 1$. W naszym rozwiązaniu jedynie ostatnia kwota może być zaokrąglona w górę, i to o maksymalnie $\lfloor \frac{G}{2} \rfloor$ groszy. Na samych pierwszych $N-1$ kwotach zaoszczędzamy $(N-1) \cdot (\lceil \frac{G}{2} \rceil - 1)$ groszy. Wobec tego w sumie zaoszczędzimy co najmniej $(N-1) \cdot (\lceil \frac{G}{2} \rceil - 1) - \lfloor \frac{G}{2} \rfloor$ groszy, co jest o co najwyżej $G-1$ mniejsze niż maksymalna możliwa zaoszczędzona kwota. Ponieważ zarówno liczba groszy zaoszczędzona przez nas jak i maksimum możliwe do zaoszczędzenia są podzielne przez G groszy, są one równe.

Liczbę groszy zaoszczędzoną przy pomocy powyższej strategii można wyrazić wzorem, co daje rozwiązanie działające w czasie $\mathcal{O}(1)$ na zapytanie.

Zadanie B

Testując różne programy, możemy wydedukować znaczenie symboli na planszy:

- `#` – blokada,
- `@` – małpa,

- `^` – punkt zbiórki,
- `)` – banan,
- `x` – zakazane pole,
- `o` oraz `0` – teleporty,

Aby rozwiązanie było poprawne, żadna z małp nie może stanąć na zakazanym polu. Jeżeli małpa spróbuje wejść w blokadę, to po prostu nie wykona żadnego ruchu. Jeżeli po wykonaniu ruchu małpa jest na polu z teleportem, to przenosi się na drugie pole z teleportem o takim samym symbolu.

Ostatni test mogliśmy rozwiązać "na kartce". Takie rozwiązanie powinno być lepszym wyborem, niż pisanie programu, bo nie zabieramy cennego czasu na komputerze innym osobom z drużyny. Jeżeli natomiast chcemy napisać program, to możemy rozwiąć to zadanie przy pomocy BFS po możliwych stanach.

Zadanie C

W zadaniu mamy dane 8 sześciennych kostek, z których każda ma pomalowane 3 sąsiadujące ze sobą wzajemnie ścianki. Celem zadania jest określenie czy można z nich złożyć większą sześcienną kostkę, która ma określone kolory na określonych ściankach (na górnej ściance kolor 1, na dolnej kolor 2, a na bokach kolejno kolory 3, 4, 5 i 6). Kostki można dowolnie ustawić i obracać.

Opiszmy kolory na kostkach jako trójki (a, b, c), gdzie przyjmując, że a jest kolorem górnej ścianki, a b prawej, to c będzie kolorem przedniej ścianki. Zatem do budowy większej kostki potrzebne nam jest 8 kostek o kolorach:

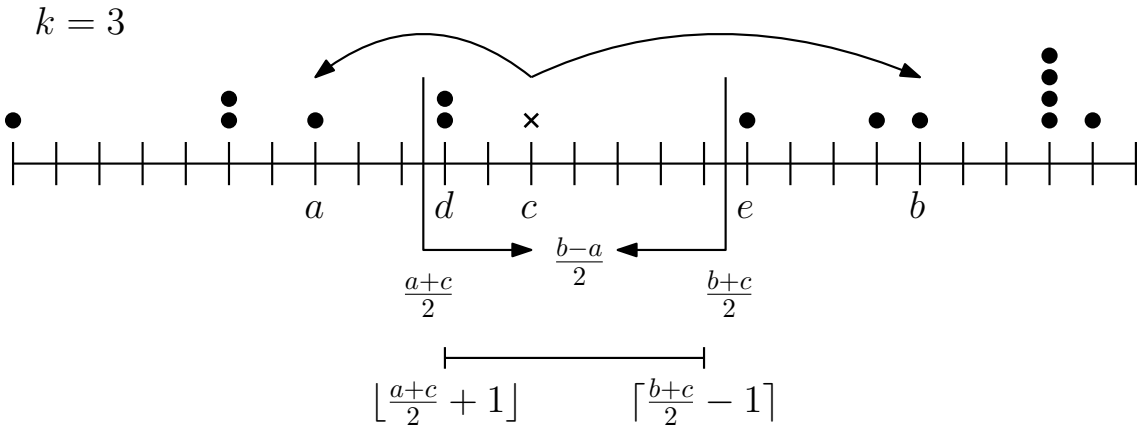
$(1, 3, 4)$, $(1, 4, 5)$, $(1, 5, 6)$, $(1, 6, 3)$, $(2, 3, 6)$, $(2, 4, 3)$, $(2, 5, 4)$, $(2, 6, 5)$.

Obrót kostki polega na tym, że przesuwamy jej kolory cyklicznie, czyli na przykład z kostki $(1, 3, 4)$ możemy otrzymać kostki $(3, 4, 1)$ albo $(4, 1, 3)$. Dodatkowo można zauważyć, że każda z powyższych 8 kostek składa się z 3 różnych kolorów, co oznacza, że nie możemy obracając jedną z nich otrzymać innej. Zatem aby rozwiązać zadanie wystarczyło dla każdej wymaganej kostki (a, b, c) (spośród tych podanych wyżej) sprawdzić czy w danych wejściowych znajduje się gdzieś kostka (a, b, c), (b, c, a) albo (c, a, b).

Zadanie D

W zadaniu mamy dane N punktów rozstawionych na osi liczbowej na pozycjach całkowitych od 0 do M. Potem my musimy wybrać pozycję, na której ustawimy swój punkt. Po tym ustawieniu ktoś wybierze pewną całkowitą pozycję na osi, która będzie *wygrywająca*, czyli k osób, które były najbliżej wygra nagrody. W razie remisów ustalamy, że my mamy pozycję przegrywającą. Zadanie polega na tym, żeby wybrać taką pozycję, która daje nam wygraną dla największej możliwej liczby pozycji wygrywających. W przypadku gdy możemy wybrać kilka pozycji, które dają nam taki wynik, mamy wybrać tą, która ma najmniejszy numer.

Na początku wyobraźmy sobie, że wybraliśmy pewną pozycję c. Niech najbliższa zajęta pozycja na lewo to d, a najbliższa zajęta pozycja na prawo to e. Oznaczmy pozycję k-tej osoby na lewo jako a, a k-tej osoby na prawo jako b.



Zauważmy, że abyśmy wygrali, to pozycja wygrywająca powinna być bliżej nas niż a i bliżej nas niż b. Zatem obszar dla nas wygrywający jest w przedziale $(\frac{a+c}{2}, \frac{b+c}{2})$. Będzie on miał albo długość $\lfloor \frac{b-a}{2} - 1 \rfloor$ albo $\lfloor \frac{b-a}{2} - 2 \rfloor$, co zależy tylko od tego czy jesteśmy na pozycji parzystej czy nieparzystej względem osób na pozycjach a i b.

Zatem całe rozwiązanie zadania polegało na tym, żeby rozważyć każdą parę osób stojących obok siebie i sprawdzić co się stanie jeśli staniemy między nimi. W tym celu znajdujemy dla nich osoby k pozycji na lewo i k pozycji na prawo, a następnie sprawdzamy jaki da nam to wynik, jeśli staniemy na najbardziej lewej pozycji wewnątrz tego przedziału oraz co jeśli staniemy na drugiej pozycji od lewej wewnątrz tego przedziału. Pozostałe pozycje dadzą nam te same wyniki, a będą ustawione badziej na prawo.

Dodatkowo należy przyjrzeć się co by się stało gdybyśmy stanęli na pozycji gdzie już ktoś stoi oraz są jeszcze dwa przypadki brzegowe – z lewej i z prawej strony. Jeden z tych przypadków to sprawdzenie co się stanie jeśli staniemy bespośrednio przed bądź dwie pozycje przed k-tą osobą od lewej strony. Drugi przypadek jest analogiczny z prawej strony.

Zadanie E

W zadaniu mamy dany stary telefon komórkowy zawierający M przycisków z różnymi cyframi (w systemie M-arnym) oraz przycisk 'backspace'. Niestety, przyciski nie maja oznaczeń, więc zanim ich nie wciśniemy, nie wiemy który przycisk jest który. Celem zadania jest określenie jaka jest oczekiwana liczba wciśnięć przycisku zanim uda nam się wybrać określony numer telefonu.

Obserwacja 1: Niezależnie od obranej strategii, każda cyfra z numeru zostanie koniec końców wciśnięta. Możemy więc w ogóle nie liczyć przyciśnięć przycisków powodujących wpisanie poprawnej cyfry, a na sam koniec dodać długość numeru do uzyskanego wyniku.

Obserwacja 2: Jan Bitovsky musi wpisywać cyfry z numeru sekwencyjnie (tzn. niezależnie od obranej strategii najpierw musi wpisać pierwszą cyfrę z numeru, następnie drugą, itd.). Jeśli dana cyfra pojawiła się już wcześniej w docelowym numerze, Jan zna już odpowiadający za nią przycisk i w optymalnej strategii po prostu go wciśnie. Możemy więc na samym początku usunąć

wszystkie powtórzenia cyfr (oprócz pierwszego). Długość numeru po tej operacji będzie nie dłuższa niż liczba różnych cyfr.

Pierwsze podejście: Spróbujmy zdefiniować, jak w dowolnym momencie może wyglądać stan wyświetlacza urządzenia. Na pierwszy rzut oka wydaje się, że potrzebujemy pamiętać zbiór wciśniętych przycisków oraz napis na wyświetlaczu. Możemy spróbować, za pomocą prostego programowania dynamicznego, dla każdego stanu policzyć wartość oczekiwaną liczby wymaganych wciśnięć do jego uzyskania, a następnie wyfiltrować spośród wszystkich możliwości te, w których Jan nie marnuje niepotrzebnie wciśnięć i zsumować ich wyniki. Niestety, liczba wszystkich poprawnych stanów jest bardzo duża – potrzebujemy lepszej reprezentacji stanu.

Co zapisywać w stanie? W rzeczywistości obserwacje pomagają nam znacznie uprościć stan urządzenia. Obserwacja 1 sprawia, że właściwie nie interesuje nas to, ile poprawnych cyfr z numeru Jan już napisał, a raczej które z nich już zlokalizował na klawiaturze (jeśli Jan w pewnym momencie będzie musiał wpisać cyfrę, której przycisk już zna, to wciśnięcie zostało już policzone w długości numeru, którą na koniec dodamy do wyniku, możemy więc ją całkowicie zignorować). Jeżeli są jakieś cyfry, które nigdy nie pojawiają się w numerze, również interesuje nas tylko liczba tych, których przycisków jeszcze nie wcisnął Jan.

Co z niepoprawnymi cyframi na wyświetlaczu? Jak tylko Jan odkryje, który z przycisków odpowiada za 'backspace', usunie cały niepoprawny sufiks. Jeżeli więc rozważamy stan, w którym Jan wpisuje złą cyfrę, możemy od razu do wyniku doliczyć koszt jej usunięcia. W ten sposób nie interesuje nas ile złych cyfr znajduje się obecnie na wyświetlaczu (koszty ich usunięcia są już doliczone), a tylko czy końcówka jest poprawna (musimy to pamiętać, gdyż Jan może przypadkowo natrafić na 'backspace' podczas posiadania poprawnego sufiksu).

Uproszczony stan: W stanie potrzebujemy więc trzymać tylko:

- liczbę pozostałych cyfr z numeru do zlokalizowania (liczba od 0 do M),
- liczbę pozostałych cyfr spoza numeru, których przycisków jeszcze nie znamy (liczba od 0 do M),
- czy przycisk 'backspace' był już raz wciśnięty (0 lub 1),
- czy sufiks napisu na wyświetlaczu jest poprawny (0 lub 1).

Takich różnych stanów będzie tylko $4 \cdot M^2$.

Rozwiązanie: Obliczamy wynik używając programowania dynamicznego. Zaczynając od stanu początkowego, przeszukujemy wszerz drzewo prawdopodobieństwa. Zauważmy, że każdy stan może znajdować się na dokładnie jednej głębokości drzewa – zawsze przechodząc między dwoma stanami Jan wciska jeden nowy przycisk, więc zmniejszy się o jeden liczba pozostałych poprawnych/niepoprawnych cyfr lub zmieni się status wciśnięcia 'backspace'. Przeszukiwanie wszerz odwiedzi więc dany stan dopiero po odwiedzeniu wszystkich stanów, które mają do niego połączenie w drzewie prawdopodobieństwa. Znając dla obecnego stanu prawdopodobieństwo znalezienia się w nim oraz wartość oczekiwaną liczby przyciśnięć możemy łatwo zaktualizować wartości wszystkich jego następników.

Zadanie F

W zadaniu rozważamy N ściśle malejących ciągów długości K. Wszystkie wartości we wszystkich ciągach są parami różne. Możemy dla uproszczenia traktować, że wartości elementów tych ciągów, tworzą permutację liczb od 1 do $N \cdot K$.

Naszym zadaniem jest znaleźć K największych wartości w tych ciągach. Jako, że K może być bardzo duże, wystarczy, że podamy pozycję ostatniej wziętej wartości w każdym ciągu.

Z tego też względu, że K jest bardzo duże, nie możemy otrzymać na wejściu wszystkich ciągów. Z tego względu możemy zadać co najwyżej 5 000 zapytań o porównanie pewnych dwóch wartości. W zadaniu $N \leq 32$, a $K \leq 2^{30}$.

W rozwiązaniu będziemy budowali rozwiązanie krok po kroku – w chwili, gdy jesteśmy pewni, że pewne elementy należą do największych, to możemy je usunąć i zmniejszyć K odpowiednio.

Kluczową obserwacją jest to, że jeśli dla pewnego V, zachodzi $K \geq V \cdot N$, to istnieje pewien ciąg z którego zabierzemy co najmniej V elementów.

Nasuwa to dość proste rozwiązanie – znajdź sensownych kandydatów na V, przykładowo potęgi dwójki, posortuj wszystkie ciągi po V-tej wartości i wybierz elementy z największego ciągu. Kontynuuj, dopóki zachodzi $K \geq V \cdot N$. Takie podejście jest z grubsza poprawne, ale wymaga doprecyzowania szczegółów.

Przede wszystkim, nie stać nas na sortowanie elementów za każdym razem, gdy chcemy znaleźć ciąg z największą wartością na V-tej pozycji. Zauważmy, że w chwili gdy zmienia się tylko jeden ciąg (ten który był maksymalny) względem reszty, to możemy zużyć tylko $\mathcal{O}(\log n)$ zapytań, aby utrzymać posortowany ciąg. Niestety, nadal zużywamy wówczas $2 \cdot N \log N \log K$ zapytań, co nie mieści się w limicie.

Zostały nam dwie obserwacje do poczynienia. Pierwsza z nich, to fakt, że nie potrzebujemy trzymać posortowany ciąg – wystarczy nam znać maksimum. W takim razie możemy użyć drzewa przedziałowego, które potrafimy zbudować używając dokładnie $N - 1$ zapytań. Redukuje to nam liczbę zapytań do $(N-1) \log K + N \log N \log K$, co jest na granicy liczby zapytań. Druga obserwacja, to fakt, że możemy ignorować potęgi dwójki, jeśli $K < V \cdot N$. Oszczędzi nam to N zapytań na konstrukcję w takim przypadku, ponieważ nie musimy budować naszej konstrukcji dla dużych potęg dwójki.

Ostatecznie, jeśli dokładnie policzymy liczbę zużywanych zapytań, to wyjdzie nam ich co najwyżej $(N-1)(\log K - \log N) + N \log N(\log K - \log N + 1) = 4935$ dla najgorszego przypadku.

Zadanie G

Dany graf dwudzielny na wierzchołkach $L \cup R$, gdzie $|L| = |R|$. Dla dowolnego zbioru $S \subseteq L$, niech $N(S)$ oznacza zbiór wszystkich sąsiadów wierzchołków z S.

Mówimy, że zbiór $S \subseteq L$ w grafie G jest *ciasny* wtw, gdy $|S| = |N(S)|$. O grafie G powiemy, że jest *dopuszczalny* wtw, gdy $\forall_{S \subseteq L} |S| \leq |N(S)|$.

Jeśli graf podany na wejściu nie jest dopuszczalny, zadaniem jest znalezienie takiego zbioru $S \subseteq L$, że $|S| > |N(S)|$. Jeżeli jednak graf jest dopuszczalny, zadaniem jest znalezienie dopuszczalnego grafu dwudzielnego G' o takim samym zbiorze wierzchołków $L \cup R$, w którym $\forall_{S \subseteq L}$ S jest ciasny w G wtw, gdy S jest ciasny w G'. Jeżeli istnieje wiele takich grafów, wybierz jeden z tych, które mają najmniejszą możliwą liczbę krawędzi.

Do zrozumienia rozwiązania potrzebne są pojęcia skojarzenia w grafie dwudzielnym, twierdzenia Halla oraz ścieżki alternującej i powiększającej.

Sprawdzanie dopuszczalności Jak wynika z twierdzenia Halla, aby zweryfikować czy graf wejściowy jest dopuszczalny wystarczy sprawdzić czy ma on pełne skojarzenie. Można użyć do tego algorytmu do znajdowania maksymalnego skojarzenia.

Przypadek gdy graf nie jest dopuszczalny Weźmy dowolny wierzchołek v, który nie został skojarzony podczas poprzedniej fazy. Znajdźmy teraz wszystkie wierzchołki, które są z niego osiągalne za pomocą ścieżek alternujących. Niech L' oznacza wierzchołki osiągalne z lewej strony, a R' wierzchołki osiągalne z prawej strony. Zauważmy, że żadna z tych ścieżek alternujących nie jest powiększająca, bo w innym przypadku moglibyśmy powiększyć nasze skojarzenie. Zatem $|L'| > |R'|$, gdyż każdemu wierzchołkowi z prawej strony odpowiada jeden skojarzony wierzchołek z lewej strony, a dodatkowo wierzchołek v nie ma skojarzenia. Dodatkowo możemy zauważyć, że R' to zbiór wierzchołków sąsiadujących z L', więc L' jest poszukiwanym zbiorem.

Przypadek, gdy graf jest dopuszczalny Weźmy znalezione pełne skojarzenie. Stwórzmy graf skierowany G na wierzchołkach z L, w którym istnieje krawędź z u do v wtw gdy istnieje ścieżka alternująca z u do v w wejściowym grafie dwudzielnym, która zaczyna się krawędzią nieskojarzoną.
RYSUNEK
Zauważmy teraz, że zbiór ciasny w wejściowym grafie to taki zbiór S w G, że wszystkie zbiory osiągalne z wierzchołków w S są również w S. Równoważnie można powiedzieć, że nie istnieje żadna krawędź z wierzchołka z S do wierzchołka, który znajduje się poza zbiorem S.

Łatwo teraz zauważyć, że dla każdego wierzchołka v z L najmniejszy ciasny zbiór, który go zawiera, to zbiór wierzchołków osiągalnych z v w grafie G. Do tego, każdy inny zbiór jest sumą ciasnych zbiorów dla pewnych wierzchołków. Zatem rozwiązaniem będie pewien graf G', w którym każdy zbiór osiągalny z każdego wierzchołka jest taki sam jak w grafie G.

Wygenerujmy graf skierowany G', który dla każdego wiezchołka v ma takie same zbiory wierzchołków osiągalnych jak pierwotny graf, ale ma najmniejszą możliwą liczbę krawędzi. Na początku rozpatrzmy wszystkie silnie spójne składowe naszego grafu. W każdej k-elementowej spójnej (dla $k > 1$) musimy mieć co najmniej k krawędzi wewnątrz niej, aby każdy wierzchołek był osiągalny z każdego innego. Dodajmy je do wyniku.

Scalmy teraz wszytkie wierzchołki z każdej spójnej składowej w jeden wierzchołek. Teraz będziemy operować na acyklicznym grafie, w którym wierzchołkami są spójne składowe. Dodając krawędzie w tym grafie spójnych składowych wystarczy, że będziemy dodawać krawędź z dowolnego wierzchołka pierwszej spójnej do dowolnego wierzchołka drugiej spójnej.

Rozważmy teraz otrzymany graf acykliczny. Przeanalizujmy wierzchołki w takiej kolejności, że każdy kolejny ma krawędzie tylko do wierzchołków, które były już analizowane. Weźmy kolejny wierzchołek v. Niech A będzie zbiorem osiągalnym z v w grafie G. Spośród wierzchołków z A niech $B \subset A$ będzie zbiorem tych wierzchołków, które nie mają żadnych krawędzi wchodzących wewnątrz A. Zauważmy, że aby wszystkie wierzchołki z A były osiągalne z v, to warunkiem koniecznym i wystarczającym jest dodanie krawędzi z v do wszystkich wierzchołków z B. Zatem dodajemy je wszystkie do zbioru wynikowego. Wykonując tę operację dla całego grafu G otrzymujemy graf G' o minimalnej liczbie krawędzi, który dla każdego wierzchołka ma taki sam zbiór elementów osiągalnych co graf G.

Implementacja Pierwsze dwie części rozwiązania można zaimplementować za pomocą algorytmu szukania maksymalnego matchingu. Ostatnia część wymaga wygenerowania grafu skierowanego (tak jak opisano powyżej). Resztę zadania można wykonać znajdując dla każdego wierzchołka jego

zbiory osiagalne, po czym analizując te wierzchołki w kolejności od tych, które mają najmniejsze zbiory, do tych które mają największe. Spójne składowe możemy znaleźć porównując które wierzchołki mają takie same zbiory wierzchołków osiągalnych. Całe to rozwiazanie można zaimplementować w czasie $O(N^3/64)$.

Generowanie grafu dwudzielnego z otrzymanego powyżej grafu skierowanego można zrobić na przykład tak, że dla każdego wierzchołka $l \in L$ dodajemu krawędź skojarzeniową $(l, l+N)$, a dla każdej krawędzi skierowanej (a, b) dodajemy krawędź $(b, a+N)$ w grafire dwudzielnym.

Zadanie H

W zadaniu mamy dane dwie długie (do 1 000 000 cyfr) liczby, składające się z cyfr 2, 3, 5 i 7. Celem zadania jest określenie czy da się sprawić aby pierwsza liczba była ściśle większa od drugiej, używając do tego co najwyżej jednej operacji zamiany dwóch dowolnych cyfr miejscami.

Przypadek gdy obie liczby są różnej długości jest prosty do rozpatrzenia, wtedy zawsze większa będzie ta liczba, która jest dłuższa (warunek ten zachodzi dlatego, że liczby nie zawierają cyfry 0). Zajmijmy się zatem sytuacją, gdy liczby mają tę samą długość.

W przypadku, gdy obie liczby używają w swoim zapisie tylko jednej i tej samej cyfry, żadna zamiana nie jest w stanie sprawić, żeby pierwsza liczba była większa od drugiej. Okazuje się, że w przeciwnym przypadku zawsze istnieje taka zamiana.

Sytuacja, gdy liczby różnią się na jakiejś pozycji też jest prosta do rozpatrzenia – wystarczy wtedy rozpatrzyć pierwszą taką pozycję i ewentualnie zamienić dane cyfry miejscami.

Ostatecznie, jeśli już obie liczby są identyczne, możemy użyć kilku sposobów na wybranie cyfr do zamiany. Jednym z nich jest znalezienie największej cyfry w drugiej liczbie i zamienienie jej z najmniejszą cyfrą w pierwszej liczbie.

Zadanie I

W zadaniu mamy dany ciąg dodatnich liczb całkowitych o wartościach do $MAX = 1\,000\,000$. Naszym zadaniem jest policzyć ile jest w nim spójnych podciągów, które zawierają trzy kolejne elementy ciągu geometrycznego, czyli elementy postaci a, $a \cdot k$ i $a \cdot k^2$, dla pewnych całkowitych wartości a i k. Elementy te mogą być w dowolnej kolejności.

Przejrzyjmy całą tablicę i dla każdego elementu x zastanówmy się co by było, gdyby to właśnie on był elementem postaci $a \cdot k^2$. W tej sytuacji k musiłoby być jego *kwardatowym dzielnikiem*, czyli takim dzielnikiem d, że liczba d^2 również jest dzielnikiem x. Zatem dla danego elementu x na pozycji i rozpatrzmy wszystkie jego kwadratowe dzielniki d i poszukajmy takich przedziałów, w których x jest elementem największym, a pozostałe dwa elementy to $\frac{x}{d}$ i $\frac{x}{d^2}$.

Niech l_1 będzie najbliższą pozycją elementu $\frac{x}{d}$ na lewo od i, a l_2 niech będzie najbliższą pozycją elementu $\frac{x}{d^2}$ na lewo od i. Podobnie na prawo, niech p_1 będzie najbliższą pozycją elementu $\frac{x}{d}$, a p_2 niech będzie najbliższą pozycją elementu $\frac{x}{d^2}$.

Rozważmy przedziały $[\min(l_1, l_2), i]$, $[l_1, p_2]$, $[l_2, p_1]$ oraz $[i, \max(p_1, p_2)]$. Zauważmy, że każdy z nich zawiera elementy x, $\frac{x}{d}$ oraz $\frac{x}{d^2}$, a dodatkowo element x jest na pozycji i. Do tego łatwo zauważyć, że każdy inny przedział zawierający te trzy elementy i element x na pozycji i musi zawierać w całości jeden z tych przedziałów. Zatem dodajmy te cztery przedziały do zbioru wynikowego.

Ostatecznie, musimy policzyć ile jest sumarycznie wszystkich przedziałów, które zawierają jeden z nich z naszych przedziałów wynikowych w całości. Można to zrobić na przykład sprawdzając dla

każdej pozycji l jaki jest najwcześniejszy koniec przedziału, który zaczyna się w l. Następnie przechodząc po wszystkich możliwych pozycjach od prawej strony można sprawdzać, gdzie byłby najbliższy dobry koniec przedziału, który zaczyna się na tej pozycji. Znając tę wartość doliczamy do wyniku taką wartość, ile jest możliwych pozycji końca przedziału na prawo od niego.

Z implementacyjnego punktu widzenia, podczas szukania kwadratowych dzielników liczby możemy pozwolić sobie na robienie tego zwykłą pętlą. Dla każdej pozycji zajmie to $O(\sqrt{MAX})$ czasu, ale są to dość szybkie operacje. W drugiej fazie, dla każdej pozycji analizujemy wszystkie kwadratowe dzielniki liczby, których może być maksymalnie 32. Elementów najbliżej na lewo i prawo możemy szukać tablicując sobie wcześniej dla każdej liczby listę jej pozycji. Wtedy rozwiązując kolejne pozycje od lewej do prawej możemy aktualizować sobie pozycje na wszystkich listach, co daje sumaryczny czas liniowy. Ostatnia faza programu jest liniowa.

Zadanie J

Rozwiązanie wzorcowe by Mateusz Orda

Wysokopoziomowa idea, próbujemy znaleźć pary sum prefiksowych takich, że zerują pierwszą połowę bitów. Następnie szukamy par par, takich że zerują również górną część bitów.

Mamy $2^{k+1}+1$ xorów prefiksowych (włącznie z pustym prefixem). Więc jeżeli patrzymy tylko na pierwszą połowę bitów (k bitów), to jest przynajmniej 2^k+1 kolizji. Niech każdy xor prefiksowy wskazuje na poprzedni xor prefikxowy o takiej samej wartości (albo nigdzie, jeżeli takiego nie ma). Zauważmy, że w ten sposób mamy dużo par o parami różnych końcach i parami różnych końcach. Ponieważ jest ich aż tyle, to wśród nich są takie dwie pary (z zasady szufladkowej), że mają takiego samego xora na górnych bitach. Otrzymujemy więc czwórkę za pomocą której wyłuskujemy rozwiązanie.

Rozwiązanie alternatywne

Z paradoksu urodzeń można powiedzieć, że jeżeli będziemy wybierali przedziały losowo, to po $\sqrt{max_val}$ losowaniach jakaś wartość xora przedziałów się powtórzy. Jeżeli przedziały nie mają wspólnego początku ani końca, to możemy wtedy od razu wypisać rozwiązanie, a jeżeli mają wspólny koniec, to dostajemy przedział o xorze równym 0. Wtedy usuwamy te przedziały z puli losowania i szukamy kolejnej kolizji.

Tym razem nawet jeżeli przedziały mają wspólny koniec to dostajemy inny przedział o xorze równym 0, więc mamy już dwa, które na pewno nie mają wspólnego końca.

Okazuje się, że rozwiązanie, które losuje cały czas szukając przedziałów bez wspólnych końców (i nadpisujące kandydatów, żeby nie utknąć w miejscu) również jest poprawne, ale dowód zostawiamy czytelnikowi jako ćwiczenie.

Zadanie K

W zadaniu mamy dany graf skierowany o N wierzchołkach i liczbę K nie mniejszą niż N^3. Należy obliczyć liczbę takich par niekoniecznie różnych wierzchołków a, b, że istnieją ścieżki z a do b i z b do a każda o długości dokładnie K.

Oczywiście a i b musza znajdować się w jednej dwuspójnej składowej, więc możemy rozważyć każdą z nich osobno. Od teraz zakładamy, że graf jest dwuspójny.

Rozpatrzmy kolorowanie wierzchołków grafu dwudzielnego na c kolorów w taki sposób, że każda krawędź biegnie od wierzchołka w kolorze i do wierzchołka w kolorze $i\%c$. Zauważmy, że takie

kolorowanie istnieje wtedy i tylko wtedy, jeśli c jest dzielnikiem wszystkich długości cykli w grafie. Rozpatrzmy dowolne takie kolorowanie. Ponieważ przejście każdej krawędzi zmienia numer koloru wierzchołka o jeden (modulo c), to wszystkie ścieżki od a do b mają taką samą długość modulo c jednoznacznie wyznaczoną przez różnicę kolorów wierzchołków a i b. Stąd wiadomo, że żeby istniała ścieżka o długości K z a do b, to $K\%c$ musi być równe różnicy kolorów a i b modulo c. Okazuje się, że jeśli c jest największym wspólnym dzielnikiem, to ten konieczny warunek jest również wystarczający.

Lemat 1. *Przedłużanie ścieżek Niech c będzie największym wspólnym dzielnikiem długości cykli w grafie o N wierzchołkach. Wtedy, jeśli $c_b = (c_a + d)\%c$, gdzie c_a i c_b oznaczają numery kolorów wierzchołków a i b, to istnieje ścieżka z a do b o długości $d + p \cdot c$ dla dowolnego $p \in \mathcal{N}$ o ile $d + p \cdot c \geq N^3$.*

Proof. Niech $l_1, l_2, \ldots, l_q$ będą wszystkimi długościami cykli prostych w grafie. Wtedy $gcd(l_1, l_2, \ldots, l_q) = c$. Z rozszerzonego algorytmu euklidesa możemy uzyskać takie całkowite współczynniki $a'_1, a'_2, \ldots a'_q$, że:

$$\sum_{i=1}^{q} a'_i \cdot l_i = c$$

Wobec tego, dla dowolnego $x \in \mathcal{N}$, istnieją takie współczynniki a_i, że:

$$\sum_{i=1}^{q} a_i \cdot l_i = x \cdot c$$

Ponadto, możemy założyć, że wszystkie a_i są większe niż $-N$. Dopóki istnieje jakieś $a_i \leq -N$ możemy wykonywać następującą procedurę zwiększającą a_i:

- Wybierzmy j takie, że $a_j > 0$ – takie musi istnieć, bo $c > 0$.
- Zwiększmy a_i o l_j – suma zwiększa się o $l_i \cdot l_j$.
- Zmniejszmy a_j o l_i – suma zwiększa się o $l_j \cdot l_i$.

Zauważmy, że nadal $a_j > -N$, bo na początku było dodatnie, a $l_i \leq N$.

Skonstruujmy następującą ścieżkę z a do b:

- b_0 razy dookoła dowolnego cyklu prostego (jego długość oznaczmy jako l_0), na którym leży a.
- Po kolei dla każdego $1 \leq i \leq N$:
  - od a do cyklu o długości l_i ścieżką o długości $e_i < N$(taka ścieżka istnieje, bo graf jest dwuspójny),
  - okrążamy ten cykl b_i razy,
  - spowrotem do a ścieżką o długości $f_i < N$.
- Od a do b dowolną ścieżką o długości $d' < N$ (taka musi istnieć, bo graf jest dwuspójny).

Oczywiście $d' \% c = d$.

Długość takiej ścieżki jest następująca:

$$d' + b_0 \cdot l_0 \sum_{i=1}^{q} b_i \cdot l_i + e_i + f_i$$

Wyznaczmy takie współczynniki, aby długość tej ścieżki była równa $d + p \cdot c$: Zauważmy, że $e_i + f_i$ jest długością cyklu, więc jest podzielne przez c, stąd niech:

$$p' := p - \sum_{i=1}^{q} \frac{e_i + f_i}{c} - \sum_{i=1}^{q} \frac{(N-1) \cdot l_i}{c} - \frac{d' - d}{c}$$

Przy prostym oszacowaniu powyższe równanie daje nam nieujemne p' dla $N \geq 4$. Dla mniejszych N dowód całego lematu jest trywialny i tutaj go pomijamy.

Następnie, ustalmy:

$$b_0 := \left\lfloor \frac{p'}{l_0} \right\rfloor$$

$$r := p' \% l_0$$

Następnie znajdujemy współczynniki a_i dla sumy dla $x = r \cdot c$ i ustalamy $b_i = (N-1) + a_i$, co zawsze jest nieujemne, ponieważ $a_i > -N$.

Tak zdefiniowana ścieżka ma długość $d + p \cdot c$, co było do wykazania. □

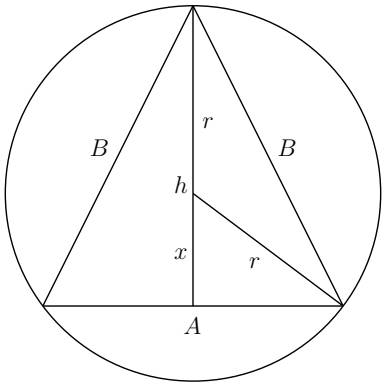
Wobec powyższego, aby znaleźć rozwiązanie należy znaleźć największy wspólny dzielnik długości cykli w grafie. Robimy to szukając kolorowań wierzchołków takich, jak wymieniliśmy powyżej – wystarczy znaleźć największe c, dla którego kolorowanie istnieje. Można to robić np. iterując się po wszystkich dzielnikach pierwszych długości dowolnego cyklu. Gdy już znaleźliśmy c to rozpatrzmy uzyskane kolorowanie w dwoch przypadkach: Jeśli $K \% c = 0$, szukamy par wierzchołków o tym samym kolorze. Jeśli $K \% c = \frac{c}{2}$, należy zliczyć pary wierzchołków o numerach kolorów różniących się o $\frac{c}{2}$. W pozostałych przypadkach żadna para wierzchołków nie jest odpowiednia.

W ten sposób uzyskujemy algorytm o złożoności $\mathcal{O}(N \log N)$.

Zadanie L

W zadaniu mamy dane okrąg o promieniu R oraz trójkąt równoramienny o podstawie długości A i ramionach długości B. Celem zadania jest określenie czy dany trójkąt zmieści się w danym okręgu.

Najbardziej naturalnym rozwiązaniem jest znalezienie promienia okręgu opisanego na danym trójkącie oraz porównanie go z promieniem podanym na wejściu. Można to zrobić na kilka sposobów (za pomocą geometrii analitycznej, z gotowego wzoru na promień okręgu opisanego itp.), ale najbardziej nasuwa się wyliczenie tego za pomocą twierdzenia Pitagorasa.



Z powyższego rysunku możemy wyliczyć kolejno, że $h = \sqrt{B^2 - A^2/4}$, $x = h - r$, a potem dla mniejszego trójkąta mamy $r^2 = A^2/4 + x^2$, z czego ostatecznie wyznaczamy $r = \frac{A^2/4+h^2}{4h}$. Tą wartość należy ostatecznie porównać z promieniem R podanym na wejściu.

Po uproszczeniu wygenerowany wzór może przyjąć na przykład taką postać:

$$B^4 < R^2(4B^2 - A^2)$$

Zadanie M

W zadaniu mamy znaleźć K-te najmniejsze leksykograficznie równanie postaci $a + b = c$, gdzie a, b i c mają ustalone (być może różne) liczby cyfr.

Skoro wyrażenie ma być najmniejsze leksyokgraficznie, to chcemy przede wszystkim zminimalizować liczbę a. Maksymalna liczba cyfr w liczbie jest równa 6, więc możemy się przeiterować po jej wszystkich możliwych wartościach. Dla ustalonego a, nierówności które musimy spełnić, żeby b miało B cyfr, a c miało C cyfr, to:

$$\begin{aligned} b &\geq 10^{B-1} \\ c = a + b &\geq 10^{C-1} \\ b &< 10^{B} \\ c = a + b &< 10^{C} \end{aligned}$$

po przerzuceniu niektórych składników na prawo otrzymujemy:

$$\begin{aligned} b &\geq 10^{B-1} \\ b &\geq 10^{C-1} - a \\ b &< 10^{B} \\ b &< 10^{C} - a \end{aligned}$$

To znaczy, że możliwe wartości b znajdują się w przedziale

$$\langle \max(10^{B-1}, 10^{C-1} - a), \min(10^B, 10^C - a) - 1 \rangle$$

.

Żeby znaleźć K-te leksykograficzne równanie, wystarczy iterować się po kolejnych a i zliczać możliwe równania.

Zadanie N

Zauważymy, że liczba robotów na lewo od wybranego robota jest równa pozycji tego robota w szeregu (licząc od 0). Zadanie sprowadza się więc do sprawdzenia, czy liczby podane na wejściu tworzą kilka ciągów arytmetycznych z różnicą równą 1 i pierwszym elementem równym 0. Niech L_i to liczba wystąpień liczb i.

Jeżeli dla jakiegoś j zachodzi $L_j < L_{j+1}$ (czyli liczb j jest mniej niż $j+1$), to nie możemy utworzyć poprawnych ciągów. W przeciwnym przypadku, dla każdego j zachodzi $L_j \geq L_{j+1}$ i możemy utworzyć poprawne ciągi.

Zrobimy to zachłannie, tworząc najdłuższe ciągi jak to tylko możliwe. Po utworzeniu ciągu wszystkie wartości L_j zmaleją o jeden, więc warunek $L_j \geq L_{j+1}$ dalej zachodzi. Część z ostatnich wartości L_j może być teraz równa zera, a wtedy kolejny utworzony ciąg będzie krótszy. Procedurę powtarzamy, aż wszystkie wartości L_j będą równe zero.

Podsumowując, w zdaniu musimy obliczyć wartości L_i – liczbę wystąpień liczby i na wejściu, a następnie sprawdzić, czy ciąg L_i jest nierosnący.

Zadanie O

Analizując pobieżnie zadanie można przekształcić je do następującej treści. Mamy daną liczbę 1 oraz operację:

- pomnóż liczbę razy a kosztem $a+1$.

Zadanie polega na znalezieniu minimalnego kosztu zamiany liczby 1 w liczbę N. Zadanie należy rozwiązać dla wielu różnych wartości N, a każda z nich może mieć wartość do $M = 1\,000\,000$.

Użyjemy programowania dynamicznego po dzielnikach. Na początku za pomocą zmodyfikowanej wersji sita Eratostenesa znajdźmy dla każdej liczby od 1 do M wszystkie jej dzielniki. Zajmie to sumaryczny czas $O(M \log M)$.

Teraz dla każdej liczby x z przedziału od 1 do M możemy policzyć jej wynik za pomocą wyznaczonych dzielników. Dokładniej, wynik to:

$$DP[x] = \min_{d \text{ takie, że } d|x} (DP[x/d] + d + 1).$$

To rozwiązanie może minimalnie nie mieścić się w czasie ze względu na duże użycie pamięci podczas trzymania dzielników. W tym celu warto zauważyć, że nie musimy ich nigdzie pamiętać, a wystarczy liczyć wartości tablicy DP podczas wykonywania sita Eratostenesa.